Can Large Language Models Write Parallel Code?

Daniel Nichols, Joshua H. Davis, Zhaojun Xie, Arjun Rajaram, Abhinav Bhave
University of Maryland
Timeline and Motivation

- **August 2021**: OpenAI Codex Release
- **November 2022**: ChatGPT Release
- **January 2023**: We train HPC-Coder [ISC24]

I wonder if these are any good at HPC?
HPC-Coder: Improving Code LLMs for HPC

Much better code generation after fine-tuning on HPC data...

but these were simple tasks.

Nichols et al., “Modeling Parallel Programs using Large Language Models”, ISC ‘24
Takeaways from HPC-Coder

LLMs are bad at Parallel Code

We prompted the LLM to write simple parallel code...

and usually got incorrect output.

We observed an ability to generate parallel code constructs, but not use them correctly
Timeline and Motivation

**August 2021**
OpenAI Codex Release

**January 2023**
We train HPC-Coder
[ISC24]

**March 2023**
GPT-4 Release

**November 2022**
ChatGPT Release

**August 2023 - Now**
StarCoder, DeepSeek, CodeLlama, …

I wonder if these are any good at HPC?

SotA Code LLM, we re-evaluate

They’re not, but we can improve them
The Need for New Benchmarks and Metrics

- **LLMs are bad with parallel code**
  - But how bad?

- **State-of-the-art changes too quickly**
  - We need a way to understand the current state of the practice

- **HPC has unique software engineering objectives**
  - Current benchmarks are mostly simple Python, Java functions
  - Performance, parallelism, efficiency are not the focus of current code LLM research
Research Questions

**RQ1** – How well do state-of-the-art LLMs generate parallel code, and which models are the best?

**RQ2** – Which parallel execution models and problem types are most challenging for LLMs?

**RQ3** – How performant and scalable is the parallel code generated by LLMs?

**RQ4** – How well can LLMs translate between execution models? How performant and scalable is the translated code?
What are code LLM benchmarks?

/* Return the value of the smallest odd number in the vector x. Use Kokkos to compute in parallel. Assume Kokkos is already initialized. Examples:

input: [7, 9, 5, 2, 8, 16, 4, 1]  
output: 1  
input: [8, 36, 7, 2, 11]  
output: 7  
*/

int smallestOdd(Kokkos::View<const int*> const& x) {
What are code LLM benchmarks?

/* Return the value of the smallest odd number in the vector x.
 Use Kokkos to compute in parallel.
 Assume Kokkos is already initialized.
 Examples:

 input: [7, 9, 5, 2, 8, 16, 4, 1]
 output: 1
 input: [8, 36, 7, 2, 11]
 output: 7
 */

int smallestOdd(Kokkos::View<const int> const& x) {
    int min_odd = std::numeric_limits<int>::max();
    Kokkos::parallel_reduce("smallestOdd", x.size(),
        KOKKOS_LAMBDA(const int i, int& update_min_odd) {
            if (x(i) % 2 != 0 && x(i) < update_min_odd) {
                update_min_odd = x(i);
            }
        },
        Kokkos::Min<int>(min_odd));
    return min_odd == std::numeric_limits<int>::max()? -1 : min_odd;
}

The LLM is given a natural language description of the problem...

and generates the code to solve that problem.

Unit tests evaluate the correctness of the generated code.
ParEval: A Parallel Code Generation Evaluation Framework

- 420 parallel code problems
  - 7 execution models
    - Serial, OpenMP, MPI, MPI+OpenMP, CUDA, HIP, Kokkos
  - 12 computational problem types
    - Sort, scan, dense linear algebra, sparse linear algebra, search, reduce, histogram, stencil, graph, geometry, fourier transform, transform
  - 5 problems per problem-type–execution model pair

- Drivers to evaluate correctness, performance, and scaling
/* Return true if `val` is only in one of vectors x or y. 
Return false if it is in both or neither. Use MPI to search in parallel. 
Assume MPI has already been initialized. 
Every rank has a complete copy of x and y. 
Return the result on rank 0. 
Examples:

input: x=[1,8,4,3,2], y=[3,4,4,1,1,7], val=7 
output: true

input: x=[1,8,4,3,2], y=[3,4,4,1,1,7], val=1 
output: false
*/

bool xorContains(std::vector<int> const& x, std::vector<int> const& y, int val) {

How do we evaluate LLM generated code?

- pass@k – How often are outputs correct?

**prompt**

/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
    int sum = 0;
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}

**LLM**

**response 1**

```c
int sum = 0;
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;
```

**response 2**

```c
int sum = 0;
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;
```

**response k**

What is the probability at least one of k responses is correct?
How do we evaluate LLM generated code?

- speedup@k – What’s the expected max speedup?

\[
\mathbb{E} \left[ \max \left\{ \frac{T_p^*}{T_{p,s_1,n}}, \ldots, \frac{T_p^*}{T_{p,s_k,n}} \right\} \right]
\]

prompt
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {
    int sum = 0;
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        sum += x[i] * y[i];
    }
    return sum;
}

LLM

response 1
int sum = 0;
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;

response 2
int sum = 0;
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;

response k

What is the expected max speedup over k responses?
How do we evaluate LLM generated code?

- speedup@k – What’s the expected max speedup?

\[
speedup_n@k = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{N} \frac{(j-1)}{(k-1)} \frac{T_p^{*}}{T_{p,j,n}}
\]

What is the expected max speedup over k responses?

prompt
/* Compute the dot product of x and y using OpenMP */
int product(int *x, int *y, size_t N) {

response 1
int sum = 0;
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;

response 2
int sum = 0;
for (int i = 0; i < N; i++) {
    sum += x[i] * y[i];
}
return sum;

response k
How do we evaluate LLM generated code?

- Correctness
  - pass@k

- Performance
  - speedup_n@k
  - speedup_max@k
  - efficiency_n@k
  - efficiency_max@k

What is the probability at least one of $k$ responses is correct?

$$\text{pass@k} = \frac{1}{|P|} \sum_{p \in P} \left[ 1 - \binom{N - c_p}{k} / \binom{N}{k} \right]$$

What is the expected max speedup over $k$ responses on $n$ resources?

$$\text{speedup}_n@k = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{N} \frac{\binom{j-1}{k-1} T_p^*}{\binom{N}{k} T_{p,j,n}}$$

What is the expected max speedup over $k$ responses on all resource counts?

$$\text{speedup}_{\text{max}}@k = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{N |\text{procs}|} \frac{\binom{j-1}{k-1} T_p^*}{\binom{N |\text{procs}|}{k} T_{p,j,n}}$$

What is the expected max efficiency over $k$ responses on $n$ resources?

$$\text{efficiency}_n@k = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{N} \frac{\binom{j-1}{k-1} T_p^*}{n \cdot T_{p,j,n}}$$

What is the expected max efficiency over $k$ responses on all resource counts?

$$\text{efficiency}_{\text{max}}@k = \frac{1}{|P|} \sum_{p \in P} \sum_{j=1}^{N |\text{procs}|} \frac{\binom{j-1}{k-1} T_p^*}{n \cdot T_{p,j,n}}$$
Choosing LLMs to Compare

- SotA code LLMs
- Open and closed source, big and small

<table>
<thead>
<tr>
<th>Model Name</th>
<th>No. of Parameters</th>
<th>Open-source Weights</th>
<th>License</th>
<th>HumanEval (pass@1)</th>
<th>MBPP (pass@1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeLlama-7B [41]</td>
<td>6.7B</td>
<td>✓</td>
<td>llama2</td>
<td>29.98</td>
<td>41.4</td>
</tr>
<tr>
<td>CodeLlama-13B [41]</td>
<td>13.0B</td>
<td>✓</td>
<td>llama2</td>
<td>35.07</td>
<td>47.0</td>
</tr>
<tr>
<td>StarCoderBase [29]</td>
<td>15.5B</td>
<td>✓</td>
<td>BigCode OpenRAIL-M</td>
<td>30.35</td>
<td>49.0</td>
</tr>
<tr>
<td>CodeLlama-34B [41]</td>
<td>32.5B</td>
<td>✓</td>
<td>llama2</td>
<td>45.11</td>
<td>55.0</td>
</tr>
<tr>
<td><strong>Phind-CodeLlama-V2 [39]</strong></td>
<td><strong>32.5B</strong></td>
<td>✓</td>
<td>llama2</td>
<td><strong>71.95</strong></td>
<td>—</td>
</tr>
<tr>
<td>GPT-3.5 [8]</td>
<td>—</td>
<td>✗</td>
<td>—</td>
<td>61.50</td>
<td>52.2</td>
</tr>
<tr>
<td>GPT-4 [34]</td>
<td>✗</td>
<td>✗</td>
<td>—</td>
<td>84.10</td>
<td>—</td>
</tr>
</tbody>
</table>

Best model at time of writing

Two popular, standard benchmarks for Python code synthesis
Using ParEval to Evaluate an LLM

420 prompts → LLM → N samples per prompt

Run samples to record correctness and performance

Use N samples to estimate metrics
**RQ1**

How well do state-of-the-art LLMs generate parallel code, and which models are the best?

All of the models are bad at writing parallel code.

Unfortunately commercial models are the best.

---

**ParEval Code Generation Results: Correctness**

<table>
<thead>
<tr>
<th>Model</th>
<th>Serial pass@1</th>
<th>Parallel pass@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL-7B</td>
<td>48.4</td>
<td>15.3</td>
</tr>
<tr>
<td>CL-13B</td>
<td>52.8</td>
<td>17.4</td>
</tr>
<tr>
<td>StarCoderBase</td>
<td>51.7</td>
<td>18.6</td>
</tr>
<tr>
<td>CL-34B</td>
<td>54.0</td>
<td>10.2</td>
</tr>
<tr>
<td>Phind-V2</td>
<td>65.6</td>
<td>32.1</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>76.0</td>
<td>39.6</td>
</tr>
<tr>
<td>GPT-4</td>
<td>76.1</td>
<td>37.8</td>
</tr>
</tbody>
</table>
**ParEval Code Generation Results: Correctness**

**RQ2**
Which parallel execution models and problem types are most challenging for LLMs?

LLMs are better for execution models “closer” to serial code.

They’re bad with distributed memory.

Small LLMs struggle with “low-data” execution models.
ParEval Code Generation Results: Correctness

RQ2
Which parallel execution models and problem types are most challenging for LLMs?

LLMs struggle with sparse, unstructured problems.

Surprisingly bad with sort and scan.
ParEval Code Generation Results: Performance

Some LLMs produce much faster code than others even when less accurate.

**RQ3**
How performant and scalable is the parallel code generated by LLMs?
Testing LLM Translation

- How well can LLMs translate between execution models?
- Translation tasks
  - Serial to OpenMP
  - Serial to MPI
  - Cuda to Kokkos
// A serial implementation of sumOfMinimumElements

/* Return the sum of the minimum value at each index of vectors x and y for all indices.
   i.e. sum = min(x_0, y_0) + min(x_1, y_1) + ...
Example:
   input: x=[3, 4, 0, 2, 3], y=[2, 5, 3, 1, 7]
   output: 10
*/

double sumOfMinimumElements(std::vector<double> const& x, std::vector<double> const& y) {
    double sum = 0.0;
    for (size_t i = 0; i < x.size(); ++i) {
        sum += std::min(x[i], y[i]);
    }
    return sum;
}

// An OpenMP implementation of sumOfMinimumElements

/* Return the sum of the minimum value at each index of vectors x and y for all indices.
   i.e. sum = min(x_0, y_0) + min(x_1, y_1) + ...
Use OpenMP to sum in parallel.
Example:
   input: x=[3, 4, 0, 2, 3], y=[2, 5, 3, 1, 7]
   output: 10
*/

double sumOfMinimumElements(std::vector<double> const& x, std::vector<double> const& y) {
...
**Translation Results**

**RQ4**

*How well can LLMs translate between execution models? How performant and scalable is the translated code?*

- Almost always better at translating than generating from scratch.
- Small LLMs benefit significantly from serial examples.
Contributions

- ParEval: a benchmark for comprehensively evaluating the ability of LLMs to generate parallel code
- Novel metrics for evaluating the performance of LLM generated code
- An in-depth study of SotA LLMs across ParEval and an identification of areas where improvement and future work is needed

Can Large Language Models Write Parallel Code?

sometimes...
The Future of ParEval

- **Adding new tests**
  - Fill-in-the-middle
  - Raja, Python (mpi4py, multiprocessing)
  - We are welcome to suggestions and contributions
  - [https://github.com/parallelcodefoundry/ParEval/](https://github.com/parallelcodefoundry/ParEval/)

- **Up-to-date dashboard**
  - [https://pssg.cs.umd.edu/blog/2024/pareval/](https://pssg.cs.umd.edu/blog/2024/pareval/)

---

“Performance-Aligned LLMs for Generating Fast Code” arXiv 2404.18864